# Ansible Patching Pipeline (End‑to‑End) — Detailed Execution + Code

**Goal:** show, step‑by‑step, how Ansible executes OS patching across many servers and reports *live* status to a Grafana dashboard using the Prometheus Pushgateway + Prometheus.

This document includes an execution timeline, the exact code you can use in a home lab (Docker/Podman compose, Prometheus config, Ansible role and playbook, templates), and comments on *vital lines* and common issues.

## Short conceptual recap (basics)

* **Ansible** runs tasks (patching commands) on remote servers via SSH. Playbooks drive the whole flow.
* **Pushgateway** is a small HTTP service that accepts metrics *pushed* by short‑lived jobs (like Ansible). It *stores* the last pushed value per grouping key until Prometheus scrapes it or the grouping is deleted.
* **Prometheus** scrapes Pushgateway on a regular interval and stores the samples in its TSDB (time‑series database).
* **Grafana** queries Prometheus and visualizes metrics in dashboards. Grafana refreshes automatically, producing the “real‑time” view.

High‑level flow:

Ansible (playbook) → Pushgateway (POST metrics during run) → Prometheus (scrapes) → Grafana (visualizes)

## Topology & components for the lab

* control01 (VM): Ansible control machine; also hosts containers for pushgateway, prometheus, grafana (compose).
* app01, db01, ... : Linux target nodes to be patched.

You can run everything in docker-compose (or podman-compose) on control01 for a compact lab.

## 1) Docker Compose (Prometheus + Pushgateway + Grafana)

**File:** docker-compose.yml

version: "3.8"  
services:  
 pushgateway:  
 image: prom/pushgateway:1.5.0  
 container\_name: pushgateway  
 ports:  
 - "9091:9091"  
 restart: unless-stopped  
  
 prometheus:  
 image: prom/prometheus:latest  
 container\_name: prometheus  
 volumes:  
 - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro  
 ports:  
 - "9090:9090"  
 depends\_on:  
 - pushgateway  
 restart: unless-stopped  
  
 grafana:  
 image: grafana/grafana:latest  
 container\_name: grafana  
 environment:  
 - GF\_SECURITY\_ADMIN\_PASSWORD=admin  
 ports:  
 - "3000:3000"  
 depends\_on:  
 - prometheus  
 restart: unless-stopped  
  
networks:  
 default:  
 driver: bridge

**Vital note:** inside this compose network containers can reach each other by name: pushgateway:9091, prometheus:9090. If you run Prometheus/Pushgateway outside compose or remote, use the absolute host address in your Ansible config.

Start the stack:

docker compose up -d  
# OR: podman-compose up -d

Open: - Grafana → http://localhost:3000 (admin/admin) - Prometheus → http://localhost:9090 - Pushgateway → http://localhost:9091/metrics (shows current groups)

## 2) Prometheus configuration

**File:** prometheus/prometheus.yml

global:  
 scrape\_interval: 5s # scrape often for near real-time view in a lab  
 evaluation\_interval: 5s  
  
scrape\_configs:  
 - job\_name: 'pushgateway'  
 static\_configs:  
 - targets: ['pushgateway:9091']  
 # NOTE: You can add relabeling rules here if needed

**Vital line:** scrape\_interval: 5s — smaller interval gives more real‑time feel but increases load. For production balance between staleness and load.

## 3) Ansible layout and approach

Project layout (recommended):

ansible/  
├── inventories/lab.ini  
├── group\_vars/all.yml  
├── patch.yml  
└── roles/patching/  
 ├── tasks/main.yml  
 └── templates/push\_metrics.j2

### Why a role?

* Encapsulates all patching logic (phase updates, retries, reboot, final push, cleanup) and can be reused.
* We will use uri to POST metrics to Pushgateway reliably (no shell‑curl dependency), add until loops for transient network problems, and use block/rescue/always so metrics are pushed even on failures.

## 4) Key design decisions & issues we must address

1. **Pushgateway stale metrics** — if you push done and do not DELETE grouping key, these metrics persist in Pushgateway. Prometheus will continue to scrape them. *Solution:* explicitly DELETE the grouping when the host finishes (or implement lifecycle cleanup via always block).
2. **Network/transient failures to Pushgateway** — if push fails, continue with patching? Usually we retry a small number of times; metric pushes are best‑effort but should not block the patch. We still attempt retries and log errors.
3. **Play interruptions** — Ansible may crash mid‑play; ensure at least the play initiates a start push early. Use always to push final status when possible.
4. **Label cardinality** — each grouping key (batch id, host, job) creates unique labels; avoid too many unique label values. Keep labels to server, env, batch and phase only.
5. **Atomic visibility** — to show phases use small numeric gauges or explicit patch\_phase{phase="patching"} gauges. Simpler: keep patch\_phase metric with multiple labels for phases and set to 1 or 0.
6. **Prometheus scrape interval** — lower interval (5s) for lab; production typically 15s or 30s. Also Grafana refresh needs to be set competitively (5–10s).

## 5) The metrics template — roles/patching/templates/push\_metrics.j2

# Exposition in Prometheus plaintext format  
# One file per push; values are overwritten when POSTed with same grouping key  
  
# Each host will set a single 'state' numeric gauge so queries are simple.  
# state: 0 = unknown / not started, 1 = start, 2 = patching, 3 = rebooting, 4 = done\_success, 5 = done\_failure  
patch\_state{server="{{ inventory\_hostname }}",env="{{ env }}",batch="{{ batch\_id }}"} {{ state | default(0) }}  
  
# reboot\_required: 1 or 0  
patch\_reboot\_required{server="{{ inventory\_hostname }}",env="{{ env }}",batch="{{ batch\_id }}"} {{ reboot\_required | default(0) }}  
  
# duration seconds (set only on final push)  
patch\_duration\_seconds{server="{{ inventory\_hostname }}",env="{{ env }}",batch="{{ batch\_id }}"} {{ duration | default(0) }}

**Vital:** Prometheus plaintext format expects metric{label="value"} <value>. We push one text body per host per update.

## 6) Ansible role tasks — roles/patching/tasks/main.yml

This is the heart of the solution. The role: 1) sets initial facts and batch id 2) pushes “start” 3) runs patch steps 4) detects reboot 5) reboots if needed 6) pushes final status and deletes grouping.

---  
- name: Patching role - main  
 hosts: all  
 gather\_facts: yes  
 become: yes  
  
 vars:  
 # default Pushgateway URL (change if not local)  
 pushgateway\_url: "http://localhost:9091"  
 patch\_job: "os\_patching"  
  
 tasks:  
  
 - name: Ensure we have a batch\_id for this run  
 # Use pipe lookup to get a timestamp string if not provided via -e  
 set\_fact:  
 batch\_id: "{{ batch\_id | default(lookup('pipe', 'date +%Y%m%d\_%H%M%S')) }}"  
  
 - name: Initialize runtime facts  
 set\_fact:  
 \_start\_epoch: "{{ ansible\_date\_time.epoch | int }}"  
 state: 1 # 1 == start  
 reboot\_required: 0  
 duration: 0  
  
 - name: Render metrics body for current state  
 set\_fact:  
 metrics\_body: "{{ lookup('template', 'push\_metrics.j2') }}"  
  
 - name: Push 'start' metrics to Pushgateway (retry on transient error)  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: POST  
 headers:  
 Content-Type: "text/plain"  
 body: "{{ metrics\_body }}"  
 status\_code: 200,202  
 register: push\_start\_resp  
 until: push\_start\_resp.status in [200,202]  
 retries: 3  
 delay: 2  
 changed\_when: false  
  
 - block:  
 - name: Set state to 'patching'  
 set\_fact:  
 state: 2  
  
 - name: Update metrics\_body for 'patching'  
 set\_fact:  
 metrics\_body: "{{ lookup('template', 'push\_metrics.j2') }}"  
  
 - name: Push 'patching' metrics (best-effort)  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: POST  
 headers:  
 Content-Type: "text/plain"  
 body: "{{ metrics\_body }}"  
 status\_code: 200,202  
 register: push\_patching\_resp  
 retries: 2  
 delay: 2  
 until: push\_patching\_resp.status in [200,202]  
 changed\_when: false  
  
 - name: Refresh package cache (Debian & RHEL families)  
 block:  
 - name: Debian: apt-get update  
 ansible.builtin.apt:  
 update\_cache: yes  
 when: ansible\_facts['os\_family'] == 'Debian'  
  
 - name: RHEL: yum/dnf makecache  
 ansible.builtin.yum:  
 update\_cache: yes  
 when: ansible\_facts['os\_family'] in ['RedHat']  
 rescue:  
 - name: If cache update fails, record failure and push  
 set\_fact:  
 state: 5 # done\_failure  
  
 - name: Apply patches (DIST upgrade on Debian, latest on RHEL)  
 block:  
 - name: Debian: perform dist-upgrade  
 ansible.builtin.apt:  
 upgrade: dist  
 register: patch\_result  
 when: ansible\_facts['os\_family'] == 'Debian'  
  
 - name: RHEL: upgrade all packages  
 ansible.builtin.yum:  
 name: '\*'  
 state: latest  
 register: patch\_result  
 when: ansible\_facts['os\_family'] in ['RedHat']  
  
 rescue:  
 - name: Patch task failed; set final failure state  
 set\_fact:  
 state: 5  
  
 - name: Update metrics\_body on failure  
 set\_fact:  
 metrics\_body: "{{ lookup('template', 'push\_metrics.j2') }}"  
  
 - name: Push failure metrics (best-effort)  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: POST  
 headers:  
 Content-Type: "text/plain"  
 body: "{{ metrics\_body }}"  
 status\_code: 200,202  
 changed\_when: false  
  
 - name: Stop further tasks for this host (we already recorded failure)  
 meta: end\_host  
  
 - name: Determine if reboot is required (Debian)  
 ansible.builtin.shell: "[ -f /var/run/reboot-required ] && echo yes || echo no"  
 register: reboot\_check  
 changed\_when: false  
 failed\_when: false  
 when: ansible\_facts['os\_family'] == 'Debian'  
  
 - name: Determine if reboot is required (RHEL using needs-restarting)  
 ansible.builtin.shell: "command -v needs-restarting >/dev/null 2>&1 && needs-restarting -r >/dev/null 2>&1 && echo yes || echo no"  
 register: reboot\_check  
 changed\_when: false  
 failed\_when: false  
 when: ansible\_facts['os\_family'] in ['RedHat']  
  
 - name: Set reboot\_required fact  
 set\_fact:  
 reboot\_required: "{{ 1 if reboot\_check.stdout is search('yes') else 0 }}"  
  
 - name: If reboot required -> mark 'reboot' state and push  
 when: reboot\_required | int == 1  
 block:  
 - name: Set state to 'rebooting'  
 set\_fact:  
 state: 3  
  
 - name: Update metrics\_body for 'reboot'  
 set\_fact:  
 metrics\_body: "{{ lookup('template', 'push\_metrics.j2') }}"  
  
 - name: Push 'reboot' metrics  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: POST  
 headers:  
 Content-Type: "text/plain"  
 body: "{{ metrics\_body }}"  
 status\_code: 200,202  
 register: push\_reboot\_resp  
 retries: 3  
 delay: 3  
 until: push\_reboot\_resp.status in [200,202]  
 changed\_when: false  
  
 - name: Reboot the server and wait for it to come back  
 ansible.builtin.reboot:  
 reboot\_timeout: 900  
 test\_command: whoami  
  
 rescue:  
 - name: If any unexpected error in block, mark failure  
 set\_fact:  
 state: 5  
  
 - name: Finalize - set final state & duration  
 set\_fact:  
 state: "{{ 4 if state | int != 5 else 5 }}" # 4==success unless previously set to 5  
 duration: "{{ (ansible\_date\_time.epoch | int) - \_start\_epoch }}"  
  
 - name: Update metrics\_body for final state  
 set\_fact:  
 metrics\_body: "{{ lookup('template', 'push\_metrics.j2') }}"  
  
 - name: Push final metrics (success/failure) - best-effort  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: POST  
 headers:  
 Content-Type: "text/plain"  
 body: "{{ metrics\_body }}"  
 status\_code: 200,202  
 register: push\_final\_resp  
 retries: 3  
 delay: 2  
 until: push\_final\_resp.status in [200,202]  
 changed\_when: false  
  
 - name: Cleanup Pushgateway grouping for this host (delete grouping)  
 ansible.builtin.uri:  
 url: "{{ pushgateway\_url }}/metrics/job/{{ patch\_job }}/instance/{{ inventory\_hostname }}/batch/{{ batch\_id }}"  
 method: DELETE  
 status\_code: 202,200  
 register: delete\_resp  
 retries: 3  
 delay: 2  
 until: delete\_resp.status in [200,202]  
 changed\_when: false

### Comments on vital lines

* set\_fact: batch\_id: ... — creates a unique batch id for the run. Keep this stable for all pushes during this run so the dashboard can filter by the batch label.
* ansible.builtin.uri ... method: POST — this is where we *push* metrics to Pushgateway in Prometheus plaintext format. Using uri avoids shell/curl and is cross-platform.
* until: push\_start\_resp.status in [200,202] + retries — retry logic for transient network errors to Pushgateway; doesn’t block indefinitely.
* register: patch\_result / rescue — capture results of the package upgrade. On failure we set state=5 (failure) and meta: end\_host to stop further tasks for that host.
* reboot step uses ansible.builtin.reboot which waits for the host to come back before continuing.
* delete with method: DELETE — IMPORTANT to avoid stale metrics persisting in Pushgateway.

## 7) Example playbook wrapper patch.yml (invocation)

---  
- name: Run patching role across lab  
 hosts: all  
 become: yes  
 vars:  
 pushgateway\_url: "http://localhost:9091" # change to the accessible URL  
 roles:  
 - role: patching

Invoke:

ansible-playbook -i inventories/lab.ini patch.yml  
# or to set explicit batch id: ansible-playbook -i inventories/lab.ini patch.yml -e "batch\_id=20250902\_150000"

## 8) Grafana queries (PromQL) — examples to build dashboard panels

* **Total targets in batch**

sum(patch\_state{batch="$batch"})

* **Count of succeeded hosts** (state==4)

sum(patch\_state{batch="$batch", state="4"})

* **Count of failed hosts** (state==5)

sum(patch\_state{batch="$batch", state="5"})

* **Hosts currently patching** (state==2)

sum(patch\_state{batch="$batch", state="2"})

* **Reboot required count**

sum(patch\_reboot\_required{batch="$batch"})

* **Average duration of finished hosts**

avg(patch\_duration\_seconds{batch="$batch", state="4"})

**Dashboard variable suggestion**: create a batch variable using label\_values(patch\_state, batch) so users can select the current run.

## 9) Walkthrough — what happens during an actual run (timeline, technical)

1. **Play start on control node**: ansible-playbook patch.yml runs. batch\_id is created and stored in facts.
2. **Per host: push “start”**: the role renders push\_metrics.j2 with state=1 and POSTs to Pushgateway grouping path /metrics/job/os\_patching/instance/<host>/batch/<batch>.
3. **Prometheus scrapes Pushgateway** at the next scrape time (within the configured scrape interval). Prometheus stores the sample (timestamped).
4. **Grafana** queries Prometheus and shows the host as Start.
5. **Role moves to state=2** and pushes patching before running the package manager upgrade. Grafana updates after Prometheus scrapes.
6. **Patching** happens (package manager); patch\_result register captures rc and output. If it fails, the rescue branch sets state=5 and pushes failure metric.
7. **Reboot check**: role runs platform specific checks to see if reboot is needed; if yes, set state=3 and push; then perform reboot. After reboot the control node reconnects and play continues.
8. **Finalization**: set state=4 on success, compute duration, push final metrics.
9. **Cleanup**: call DELETE on the Pushgateway grouping path — this removes the current grouping from Pushgateway so it doesn’t show as active there. Prometheus will still have stored the time‑series data for historical queries.

## 10) Troubleshooting & common pitfalls (detailed)

* **Pushgateway shows stale entries**: If grouping keys are never deleted (or deletes fail), Pushgateway keeps them and Prometheus will keep scraping them. Always attempt DELETE in the always/cleanup phase.
* **Metrics missing in Prometheus**: check prometheus UI → Status → Targets to ensure pushgateway:9091 is UP. Also inspect /metrics on the Pushgateway in browser to ensure pushes happened.
* **Ansible cannot reach Pushgateway**: if Ansible runs on a control host outside Docker network, change pushgateway\_url to http://<host-ip>:9091 or http://host.docker.internal:9091 (platform specific).
* **Large cardinality**: avoid labels such as package\_name or per‑package fields. Keep labels limited; batch is OK but avoid per‑host millions of unique labels.
* **Play stops before final push**: use block/rescue/always to guarantee some metrics are pushed even on exceptions. Always push start early to show attempts.
* **Prometheus scrape interval too long**: increase scrape frequency in dev/lab for a real‑time feel (e.g., 5s) but be mindful of load in production.

## 11) Testing & validation steps (quick checklist)

1. docker compose up -d and confirm services are running.
2. Visit http://localhost:9091/metrics and verify it’s accessible.
3. Run ansible-playbook -i inventories/lab.ini patch.yml -e "batch\_id=test\_001" on a single target.
4. Use curl http://localhost:9090/api/v1/query?query=patch\_state to see time series returned.
5. Import or create Grafana dashboards and add the batch variable. Observe panel updates while the playbook runs.

## 12) What to show on slides (short list)

* Slide 1: Title + flow diagram (Ansible → Pushgateway → Prometheus → Grafana). 1 line per component describing role.
* Slide 2: Patch phases & state mapping (0..5 numeric mapping table) + why grouping key (batch) matters.
* Slide 3: Code snippet (Ansible uri POST + grouping URL) + explanation of vital lines.
* Slide 4: Prometheus scrape & Grafana refresh explanation (timing & delays).
* Slide 5: Pitfalls & mitigation (stale metrics, retries, label cardinality).
* Slide 6: Demo checklist & how to run locally.

## 13) Next steps I can do for you (pick one)

* Produce a **ZIP** containing the compose files, Prometheus config, the Ansible role and a starter Grafana dashboard JSON ready to import. (I can generate this here.)
* Generate a **PowerPoint** (.pptx) from the slides outline above with the key code snippets included.
* Produce a minimal **Grafana dashboard JSON** (I can attach it in the zip) ready for import.

If you’d like the ZIP or PPTX, tell me which one and which OS family you want to target (RHEL family or Debian family or both) — I can package the code accordingly.